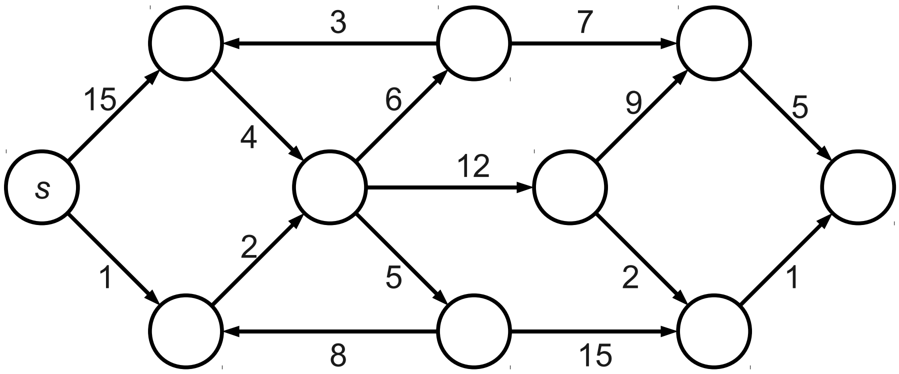
Corso di Algoritmi e Strutture di Dati

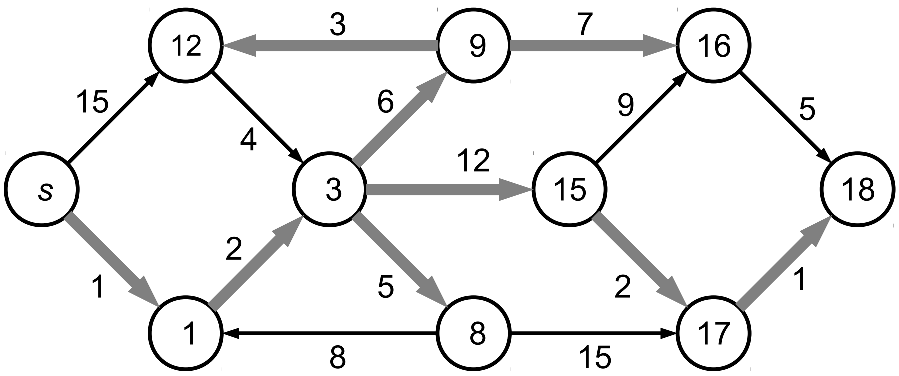
Esercizi

Esercizio 1. Si consideri il grafo orientato G=(V, E), ai cui archi sono associati costi positivi come illustrato in figura:



Applicare "manualmente" l'algoritmo di Dijkstra per calcolare un albero dei cammini di costo minimo partendo dal nodo sorgente s (è il nodo più a sinistra in figura). Mostrare il risultato finale dell'algoritmo di Dijkstra, inserendo all'interno di ogni nodo la distanza minima da s, ed evidenziando opportunamente (ad esempio, cerchiando il valore del costo) gli archi che fanno parte dell'albero dei cammini di costo minimo.

Soluzione.



Esercizio 2. Si consideri un grafo non orientato $G = (V, E)$, non necessariamente connesso, in cui tutti gli archi abbiano lo stesso peso positivo $\alpha > 0$.

1. Scrivere un algoritmo efficiente che, dato in input il grafo G di cui sopra e due nodi u e v, calcola la lunghezza del cammino di costo minimo che collega u e v. Se i nodi non sono connessi, l'algoritmo restituisce $+\infty$.
2. Analizzare il costo asintotico dell'algoritmo proposto.

Soluzione. È ovviamente possibile usare l'algoritmo di Dijkstra per il calcolo dell'albero dei cammini di costo minimo. Tuttavia, una tale soluzione NON è la più efficiente in questo caso: una soluzione con costo computazionale inferiore consiste nell'esplorare il grafo a partire dal vertice u utilizzando una visita in ampiezza (BFS, Breadth First Search). Ricordiamo che la visita BFS esplora i nodi in ordine non decrescente rispetto alla distanza (intesa come numero minimo di archi) dal nodo sorgente. Nel nostro caso, poiché tutti gli archi hanno lo stesso peso positivo α, la minima distanza di v da u è data dal minimo numero di archi che separa i due nodi, come determinato dall'algoritmo BFS, moltiplicato per α.

Lo pseudocodice seguente realizza una versione di BFS semplificata, in cui (i) non viene mantenuto esplicitamente l'albero di copertura (in quanto non richiesto dall'esercizio), e (ii) si utilizza l'attributo *v*.dist settato a +∞ per denotare i nodi che non sono ancora stati esplorati (in modo tale da evitare l'ulteriore attributo booleano *v*.mark).
Lo pseudocodice è il seguente:

```
DistanzaMinimaAlpha( grafo G = (V, E), nodo u, nodo v real α ) → real
  // inizializza ogni nodo w come inesplorato
  foreach w in V do
      w.dist ← +∞;
  endfor;
  Queue Q;
  u.dist := 0;
  Q.enqueue(u);
  while ( not Q.isEmpty() ) do
      nodo n ← Q.dequeue();
      if (n == v) then
          return n.dist * α; // abbiamo raggiunto v
      endfor
      foreach w adiacente a n do
          if (w.dist == +∞) then
              w.dist ← n.dist + 1;
              Q.enqueue(w);
          endif
      endfor
  endwhile
  // se siamo arrivati qui, significa che il nodo v non è stato raggiunto
  return +∞;
```

L'algoritmo esegue una visita BFS dell'intero grafo nel caso peggiore, per cui il costo computazionale è lo stesso della visita BFS, ossia $O(n + m)$ (n = numero di nodi, m = numero di archi, se il grafo viene implementato tramite liste di adiacenza).

Esercizio 3. Una rete stradale è descritta da un grafo orientato pesato $G = (V, E, w)$. Per ogni arco (u, v), la funzione costo $w(u, v)$ indica la quantità di carburante (in litri) che è necessario consumare per percorrere la strada che va da u a v. Tutti i costi sono strettamente positivi. Un veicolo ha il serbatoio in grado di contenere C litri di carburante, inizialmente completamente pieno. Non sono presenti distributori di carburante.
Scrivere un algoritmo efficiente che, dati in input il grafo pesato G, la quantità C di carburante inizialmente presente nel serbatoio, e due nodi s e d, restituisce *true* se e solo se esiste un cammino che consente al veicolo di raggiungere d partendo da s, senza esaurire il carburante durante il tragitto.

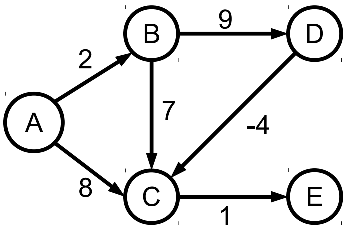
Soluzione. È sufficiente eseguire l'algoritmo di Dijkstra e calcolare il vettore $D[v]$ delle distanze minime dalla sorgente s a ogni nodo v raggiungibile da s, utilizzando il consumo di carburante come peso. È possibile interrompere l'esecuzione dell'algoritmo di Dijkstra appena si raggiunge d oppure si supera il costo C. L'algoritmo restituisce true se e solo se si raggiunge d e non si è superato C.

Esercizio 4. Si consideri un grafo orientato pesato, composto dai nodi $\{A, B, C, D, E\}$, la cui matrice di adiacenza è la seguente (le caselle vuote indicano l'assenza dell'arco corrispondente; l'intestazione di ogni riga indica il nodo sorgente, mentre l'intestazione della colonna indica il nodo destinazione):

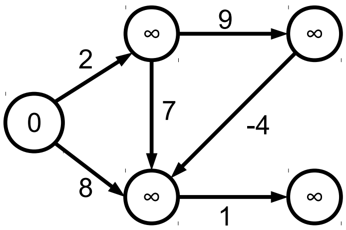
	A	***B***	***C***	***D***	***E***
A		2	8		
B			7	9	
C					1
D			-4		
E					

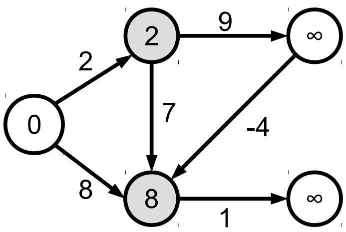
2. Determinare la distanza minima di ciascun nodo dal nodo sorgente *A*. Quale degli algoritmi visti a lezione puo' essere impiegato?

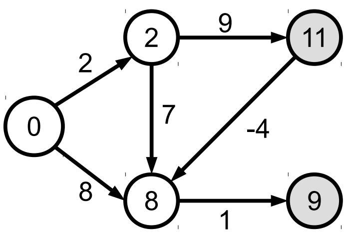
Soluzione. Il grafo è il seguente

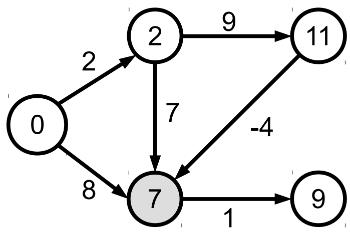


Si nota come l'arco (D, C) abbia peso negativo; quindi per calcolare le distanze minime non è possibile usare l'algoritmo di Dijkstra, e si deve ricorrere ad esempio all'algoritmo di Bellman-Ford. Le iterazioni dell'algoritmo sono indicate nel seguito (i nodi in grigio sono quelli la cui distanza cambia):

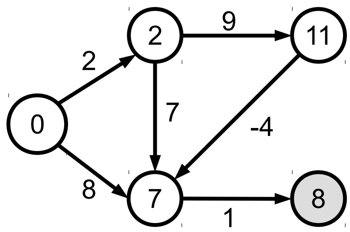








0
2
7
11
9
2
8
9
7
-4
1



0
2
7
11
8
2
8
9
7
-4
1